Elementary linear logic revisited for polynomial time and an exponential time hierarchy (extended version)

Patrick Baillot

ENS Lyon, Universite de Lyon, LIP (UMR 5668 CNRS-ENSL-INRIA-UCBL)
patrick.baillot@ens-lyon.fr

Abstract. Elementary linear logic is a simple variant of linear logic, introduced by Girard and which characterizes in the proofs-as-programs approach the class of elementary functions (computable in time bounded by a tower of exponentials of fixed height). Other systems, like light linear logic have then been defined to capture in a similar way polynomial time functions, but at the price of either a more complicated syntax or of more involved encodings. Such logical systems can then be the basis of type systems to guarantee statically complexity properties on lambda-calculus.
Our goal here is to show that despite its simplicity, elementary linear logic can nevertheless be used as a common framework to characterize the different levels of a hierarchy of deterministic time complexity classes, within elementary time. We consider a variant of this logic with type fixpoints and weakening (elementary affine logic with fixpoints). The key ingredients are then the choice of specific types and a finer study of the normalization procedure on proof-nets. We characterize in this way the class P of polynomial time predicates and more generally the hierarchy of classes $\mathsf{k\text{-}EXP}$, for $k \geq 0$, where $\mathsf{k\text{-}EXP}$ is the union of $\mathsf{DTIME}(2_k^{n^i})$, for $i \geq 1$.

1 Introduction

Implicit computational complexity. This line of research promotes investigations to delineate classical complexity classes by programming languages or logics, without refering to explicit bounds on resources (time, space ...) but instead by restricting the primitives or the features of the languages. Various approaches have been used for that, primarily in logic and in functional languages: restrictions of the comprehension scheme in second-order logic [Lei91,Lei02]; ramification in logic or in recursion [Lei94,BC92]; *read-only* functional programs [Jon01]; variants of linear logic [Gir98,Laf04]... to name only a few.

Note that the programming disciplines induced by these systems are quite restrictive, but some of these characterizations have in a second step led to more flexible criteria for statically checking complexity bounds on programs:

ramification and safe recursion have inspired the work on interpretation methods for complexity [BMM11] on the one-hand, and linear type systems for non-size-increasing computation [Hof03] on the other, which itself has led to typing methods for amortized complexity analysis [HJ06,HAH11].

Linear logic. The linear logic approach to implicit complexity fits in the proofs-as-programs paradigm. It stems from the observation that as duplication is controlled in linear logic by the modality !, weaker versions of this modality can define systems with a complexity-bounded normalization procedure: *elementary linear logic* (ELL) [Gir98,DJ03] characterizes in this way the class of Kalmar elementary functions (computable in time bounded by a tower of exponentials of fixed height), while light linear logic (LLL) [Gir98] and soft linear logic (SLL) [Laf04] characterize functions computable in polynomial time. These logical systems have then enabled the design of type systems for λ-calculus or functional languages ensuring that a well-typed program has a polynomial time complexity bound [BT09,GR07,BGM10].

Note that initially ELL does not sound as interesting as LLL and SLL since it corresponds to elementary complexity, which is not very relevant from a programming point of view. However it has nice logical properties, a simpler language of formulas than LLL (no § modality) and allows for a more natural programming style than SLL. It has also been studied for its remarkable properties concerning λ-calculus optimal reduction [ACM04].

Calibrating complexity. One might however deplore a lack of homogeneity in these characterizations of complexity classes by variants of linear logic: indeed some common deterministic complexity classes like EXP have not yet been characterized, a different system is needed for each complexity class, and these various systems are not easy to compare.

By contrast, other methods in implicit complexity have provided frameworks that can be calibrated to delineate different complexity classes inside the (large) Kalmar elementary class:

- Jones considers in [Jon01] a *read-only* functional programming language and characterizes in it the classes $kexptimek$, for $k \geq 0$, by considering, for each k, programs using only arguments of type-order at most k;
- Leivant investigates in [Lei02] second-order logic with comprehension (quantifier elimination) restricted to various families of first-order formulas: the functions provably total in this logic with comprehension restricted to formulas of type-order at most k are precisely the functions of $\mathsf{k\text{-}EXP}$.

Note that even if these two frameworks fit in different computational approaches, they both use as parameter the type-order (or implicational rank) of formulas.

Objective and contribution. A goal of the present work is to provide an analogous framework in linear logic, allowing to characterize in a single logic a hierarchy of complexity classes by calibrating a certain parameter. We will use elementary linear logic, which offers the advantage of simplicity. A key parameter in this system, as in LLL, is the number of nested modalities (!) and this will be the value calibrating our complexity bounds.

For technical reasons we will actually consider an extension of ELL obtained by adding to it type fixpoints. It had been observed from the beginning [Gir98] that this feature does not modify the dynamics of ELL and its complexity bounds. We will also allow unrestricted weakening, which is innocuous and common practise since [AR02]. Given some types W for binary words and B for booleans, we will then show that for $k \geq 0$, the proofs of conclusion $!W \multimap !^{k+2}B$, where $!^i$ stands for a sequence of i !s, correspond to the predicates of k-EXP. In particular this gives, in the case where $k = 0$, a characterization of the class P in an elementary logic, with the type $!W \multimap !^2B$.

Note that a distinctive point of our characterization is that it does not rely on a restriction of a particular operation *inside* the proof-program, like the application of a function to an argument in [Jon01] or the comprehension rule in [Lei02]. Instead it only imposes a condition on the *conclusion* of the proof (or type of the program), that its to say on its interface. In this sense it is more modular than these previous characterizations. Observe also that our system is a second-order logic, as the one of [Lei02], but here comprehension is not restricted.

The main technical tool we will use to establish our results is that of proof-nets, a graphical representation of proofs which allows for a fine study of normalization. They were already used in [Gir98] but we will here carry out a finer analysis of normalization in the particular cases of the proof-nets with the above-mentioned conclusions, which will lead to sharper complexity bounds.

2 Characterization of the classes P and k-EXP

We consider intuitionistic affine elementary logic with type fixpoints that we denote by EAL_μ. Actually for our purpose it is sufficient to consider its multiplicative fragment. The grammar of types is:

$$A ::= \alpha \mid A \multimap A \mid A \otimes A \mid !A \mid \forall \alpha.A \mid \mu\alpha.A$$

We will represent functions by proofs, but as often it will be convenient to use λ-calculus to denote the algorithmic content of proofs. For that we will consider an extension of λ-calculus with a $\otimes$ construction:

$$t, u ::= x \mid \lambda x.t \mid (t\ u) \mid t \otimes u \mid \mathrm{let}\, t\, \mathrm{be}\, x \otimes y\, \mathrm{in}\, u$$

Its reduction rule is obtained by the context-closure of the usual β-reduction rule and of the following one:

$$\mathrm{let}\, t_1 \otimes t_2\, \mathrm{be}\, x \otimes y\, \mathrm{in}\, u \to u[t_1/x, t_2/y].$$

The rules of the EAL_μ, are now given on Fig. 1, as a sequent calculus decorated with λ-terms. This system only differs from the intuitionistic version of elementary linear logic without additive connectives [Gir98,DJ03] by the fact that we have added the fixpoint construction (rules L_μ and R_μ) and allowed for general weakening (rule (Weak)). Observe that some rules do not have any effect on the

term (!, L_μ, R_μ, $L_\forall$, $R_\forall$): this is because we want to keep the term calculus as simple as possible, and the current calculus is anyway sufficient to represent the functions denoted by the proofs.

Observe that the formulas $!A \multimap A$ and $!A \multimap !!A$ are not provable, which is the distinctive feature of elementary linear logic with respect to ordinary linear logic.

Finally, note that if we added the fixpoint rules to intuitionistic logic or linear logic cut elimination would not be normalizing anymore, but strong normalization does hold for EAL_μ (see [Gir98]).

Axiom and Cut.

$$\frac{}{x : A \vdash x : A}\ Ax \qquad \frac{\Gamma \vdash t : A \quad \Delta, x : A \vdash u : B}{\Gamma, \Delta \vdash u[t/x] : B}\ Cut$$

Structural Rules.

$$\frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash t : A}\ Weak \qquad \frac{\Gamma, x_1 :!A, x_2 :!A \vdash t : B}{\Gamma, x :!A \vdash t[x/x_1, x/x_2] : B}\ Contr$$

Multiplicative Rules.

$$\frac{\Gamma \vdash t : A \quad \Delta, x : B \vdash u : C}{\Gamma, \Delta, y : A \multimap B \vdash u[(y\ t)/x] : C}\ L_{\multimap} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ R_{\multimap}$$

$$\frac{\Gamma, x_1 : A, x_2 : B \vdash t : C}{\Gamma, x : A \otimes B \vdash \mathrm{let}\, x\, \mathrm{be}\, x_1 \otimes x_2\, \mathrm{in}\, t : C}\ L_{\otimes} \qquad \frac{\Gamma \vdash t_1 : A \quad \Delta \vdash t_2 : B}{\Gamma, \Delta \vdash t_1 \otimes t_2 : A \otimes B}\ R_{\otimes}$$

Exponential Logical Rule.

$$\frac{\Gamma \vdash t : A}{!\Gamma \vdash t :!A}\ !$$

Second Order Rules

$$\frac{\Gamma, x : C[A/\alpha] \vdash t : B}{\Gamma, x : \forall\alpha.C \vdash t : B}\ L_\forall \qquad \frac{\Gamma \vdash t : C \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash t : \forall\alpha.C}\ R_\forall$$

Fixpoint Rules

$$\frac{\Gamma, x : A[\mu\alpha.A/\alpha] \vdash t : B}{\Gamma, x : \mu\alpha.A \vdash t : B}\ L_\mu \qquad \frac{\Gamma \vdash t : A[\mu\alpha.A/\alpha]}{\Gamma \vdash t : \mu\alpha.A}\ R_\mu$$

Fig. 1. The system EAL_μ

Let us denote the following types respectively for booleans, n-ary finite types, tally integers and binary words, and which are adapted from system F:

$$\begin{aligned}
\mathbf{B} &= \forall\alpha.\alpha \multimap \alpha \multimap \alpha \\
\mathbf{B^n} &= \forall\alpha.\alpha \multimap \ldots \multimap \alpha, \text{ with } n+1 \text{ occurrences of } \alpha \\
\mathbf{N} &= \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \\
\mathbf{W} &= \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)
\end{aligned}$$

Recall that these data-types admit some coercions [Gir98]. For $\mathbf{W}$ for instance, one can give a proof of type $\mathbf{W} \multimap !\mathbf{W}$ which, as a λ-term, acts as an identity on the terms encoding binary words.

We denote: $2_0^n = n$, $2_{k+1}^n = 2^{2_k^n}$. Recall that elementary functions are the functions computable on a Turing machine in time $O(2_k^n)$, for some k.

The standard results of elementary linear logic [DJ03] still hold in the setting of EAL_μ:

- for any integer d, any proof of $\mathbf{N} \multimap !^d\mathbf{N}$ represents an elementary function (*complexity soundness*);
- for any elementary function $f : \mathbb{N} \to \mathbb{N}$ there exists an integer d and a proof of $\mathbf{N} \multimap !^d\mathbf{N}$ representing it (*extensional completeness*).

Now, by considering alternative types we can delineate more complexity classes:

Theorem 1. *We consider the system EAL_μ.*

- *The functions representable by proofs of $!\mathbf{W} \multimap !^2\mathbf{B}$ (resp. $!\mathbf{W} \multimap !^3\mathbf{B}$) are exactly the class* P *(resp.* EXP*);*
- *More generally, for any $k \geq 0$, the functions representable by proofs of $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$ are exactly the class* $\mathsf{k\text{-}EXP}$*.*

where:
$\mathsf{k\text{-}EXP} = \cup_{i\in\mathbb{N}}\mathsf{DTIME}(2_k^{n^i})$,
$\mathsf{EXP} = \mathsf{1\text{-}EXP}$.

Remark 1. Note that we do not use fixpoints in the final types involved. However, technically speaking the fixpoints are used in the proofs of completeness, in order to simulate polynomial time (resp. k-exponential time) Turing machines, as we will see in Sect. 4.

Remark 2. With the type $!\mathbf{W} \multimap !\mathbf{B}$ one characterizes only the constant functions.

Observe that as in [Jon01] we are characterizing here predicates and not general functions. We will come back to this point later, in Remark 6.

In the rest of the paper we will prove Theorem 1: Sect. 3 will establish the complexity soundness (proofs with these types represent predicates of the given complexity classes) and Sect. 4 will prove the extensional completeness (all predicates of these complexity classes can be computed by suitable proofs).

3 Proof-nets and complexity soundness

In order to prove the complexity bound we study the cut elimination process and take advantage of the assumption that the conclusion of the proof is of type $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$ in order to derive a sharper bound. For that, as usual in linear logic it is convenient to use *proof-nets* to analyse cut elimination as proof-net reduction. The proof-nets will use formulas of classical elementary linear logic with fixpoints:

$$A ::= \alpha \mid \alpha^{\perp} \mid A \otimes A \mid A \wp A \mid !A \mid ?A \mid \mu\alpha.A \mid \overline{\mu}\alpha.A \mid \forall\alpha.A \mid \exists\alpha.A$$

The connectives (modalities) ! and ? are called *exponentials* and $\otimes/\wp$ are *multiplicatives*.

Formulas of EAL_μ are translated in this grammar by using $A \multimap B \equiv A^{\perp} \wp B$ and the usual linear logic De Morgan laws for linear negation:

$$\begin{array}{ll} (A \otimes B)^{\perp} \equiv A^{\perp} \wp B^{\perp}, & (!A)^{\perp} \equiv ?A^{\perp}, \\ (\mu\alpha.A)^{\perp} \equiv \overline{\mu}\alpha.A^{\perp}, & (\forall\alpha.A)^{\perp} \equiv \exists\alpha.A^{\perp}, \qquad A^{\perp\perp} \equiv A. \end{array}$$

In order to handle weakening we will use proof-nets with polarities, following [AR02]. Note that proof-nets with polarities had been considered before, e.g. in [Lam96], but here we will follow the conventions and notations of [AR02].

The nodes are described on Fig. 2:

- nodes have ports which are positive (dark bullet) or negative (white bullet); an edge can link together either two positive or two negative ports; we say that an edge is positive (resp. negative) if it is connected to a positive (resp. negative) port;
- as the system is affine, the proof-nets use, beside the weakening w-node, also an h-node;
- two nodes μ and $\overline{\mu}$ are added, corresponding resp. to the fixpoint rules R_μ and L_μ ;
- each node ! or ? comes with a !-box, as shown on Fig. 3; two boxes are either disjoint or one is included in the other; the !-node is called the *principal door* of the box and the ?-nodes are its *auxilliary doors*.

One will translate an EAL_μ proof π of conclusion $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ by a graph π^* with conclusions ${A_1}^{\perp}, \ldots, {A_n}^{\perp}, B$ where the edges of the $A_i^{\perp}$ $(1 \leq i \leq n)$ are negative and the edge of B is positive. This translation is standard [Gir87] and we only describe a couple of illustrative cases:

- if π is obtained by an Ax rule, then π^* is an ax-node;
- if π is obtained by a Cut rule between π_1 and π_2, then π^* is obtained by linking ${\pi_1}^*$ and ${\pi_2}^*$ by a cut-node;
- if π is obtained by a !-rule on π_1, then π^* is obtained by applying a !-box on ${\pi_1}^*$ (see Fig. 3),

and so on. On Fig. 2 below each node we have indicated the sequent-calculus rule it corresponds to. Note that the h-node is not used for this translation; it will only appear during normalisation.

Now, a graph R is called a *proof-net* if there exists a proof π such that $R = \pi^*$.

A *cut*-node between an ax-node (resp. w-node) and another node (resp. another node which is not an ax-node) is called an *axiom cut* (resp. a *weakening cut*). A cut between an $\otimes$-node and a $\wp$-node is called a *multiplicative cut*. A cut between a !-node and a ?-node or ?c-node (resp. a ?c-node) is called an *exponential cut* (resp. a contraction cut). *Quantifier cuts* and μ*-cuts* are defined in an analogous way.

In a proof-net, a maximal tree with ?-nodes, ax-nodes and w-nodes (of type $?A^{\perp}$) as leaves, and ?c-nodes as internal nodes, is called an *exponential tree*.

Now, given R, the *depth* of a node is the number of exponential boxes containing it, and the depth $d(R)$ of R is the maximal depth of its nodes.

Let $|R|_i$ denote the number of nodes at depth i which are not cut nodes, w-nodes or h-nodes. We denote $|R| = \Sigma_{i=0}^{d(R)} |R|_i$.

Reduction. We can describe the reduction procedure on proof-nets, which consists in eliminating cuts. It is defined by the rules given on Fig. 4, 5, 6, 7:

- Fig. 5 shows the reduction of exponential cuts (contraction step and box-box step);
- Fig. 6 shows the reduction of weakening cuts; notice that one of these steps introduces h-nodes;
- Fig. 7 shows the reduction of cuts on h-nodes.

Observe that during a reduction step the depth of an edge does not change, hence the depth of the proof-net does not increase. This is called the *stratification property* [DJ03] and it is a key ingredient for the complexity properties. It is not valid in ordinary linear logic, and it comes from the fact that during reduction a !-box is not opened and does not enter another box (see Fig. 5).

We say that an exponential cut c is a *special cut* if the box $\mathcal{B}$ corresponding to the ! node does not have any cut below its auxilliary ports. The following fact can be easily verified by examining each reduction step other than the contraction reduction step:

Lemma 1. *Let R be a proof-net and R' be obtained from R by reducing a cut at depth i which is not a contraction cut. Then we have $|R'|_i < |R|_i$ and $|R'| < |R|$.*

Now let us briefly recall the method used in [Gir98] to establish complexity bounds on proof-net reduction in light linear logic and elementary linear logic. It uses a specific reduction strategy, obtained by the two following ideas:

- reduce the cuts level-by-level, that is to say first at depth i (round i) for i successively equal to $0, 1, \ldots, d(R)$;
- at a given depth i, proceed in two phases:
  - first reduce cuts that make the size decrease, so in the case of ELL all cuts but the exponential cuts,

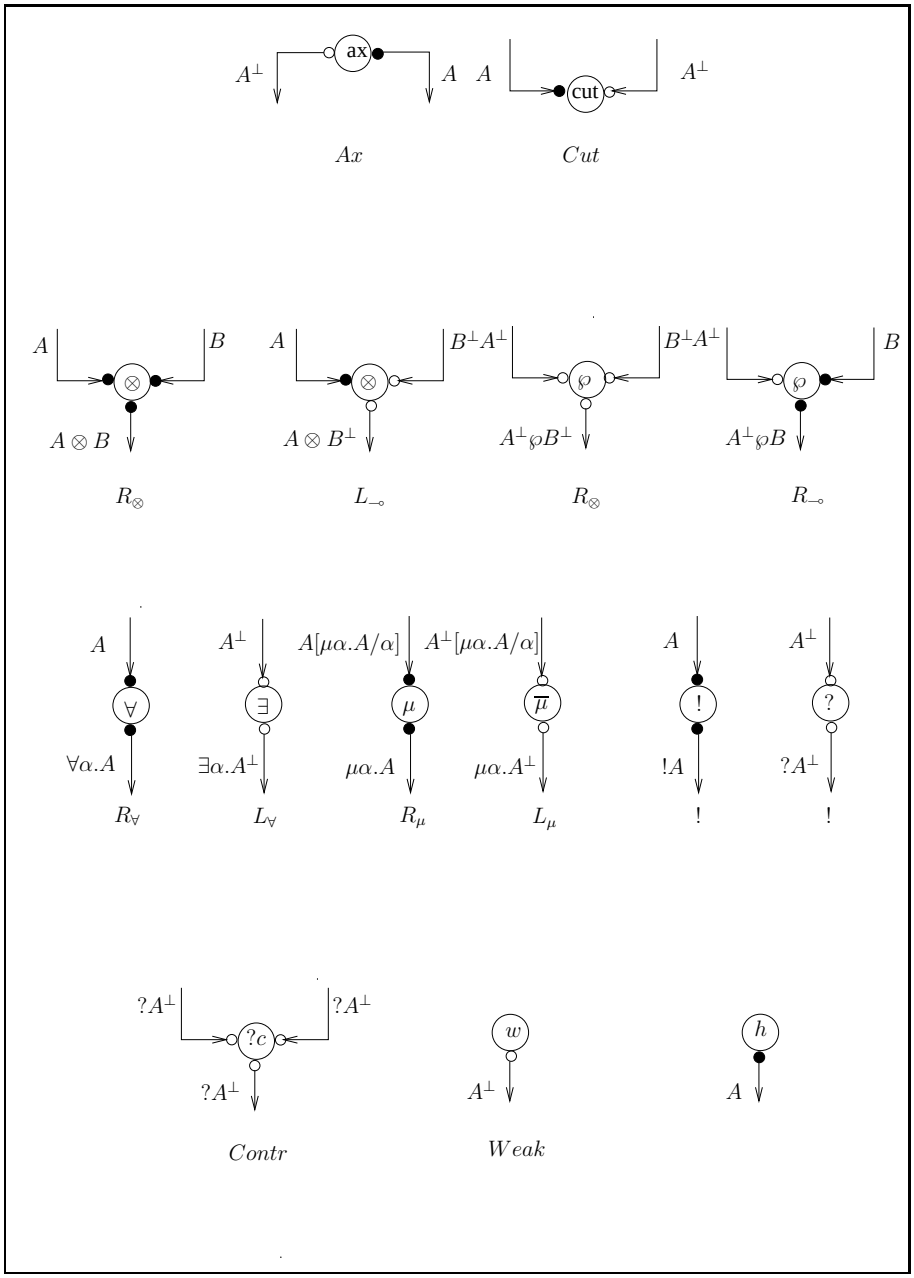

Fig. 2. Nodes of the proof-nets.

- then reduce the exponential cuts, by repeatedly reducing a *special cut*.

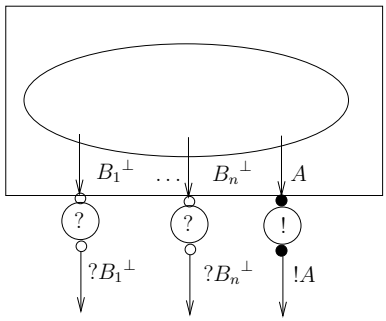

Fig. 3. !-box.

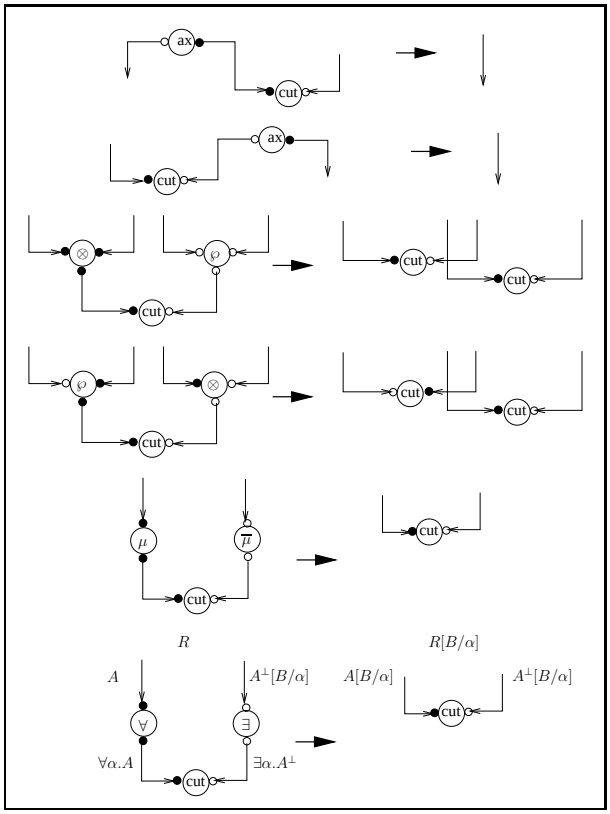

Fig. 4. Reduction steps (1/4).

Let us denote by R^i the proof-net at the beginning of round i. In order to bound the number of steps of this reduction strategy, the proof proceeds by, for $i = 0$ to $d(R)$:

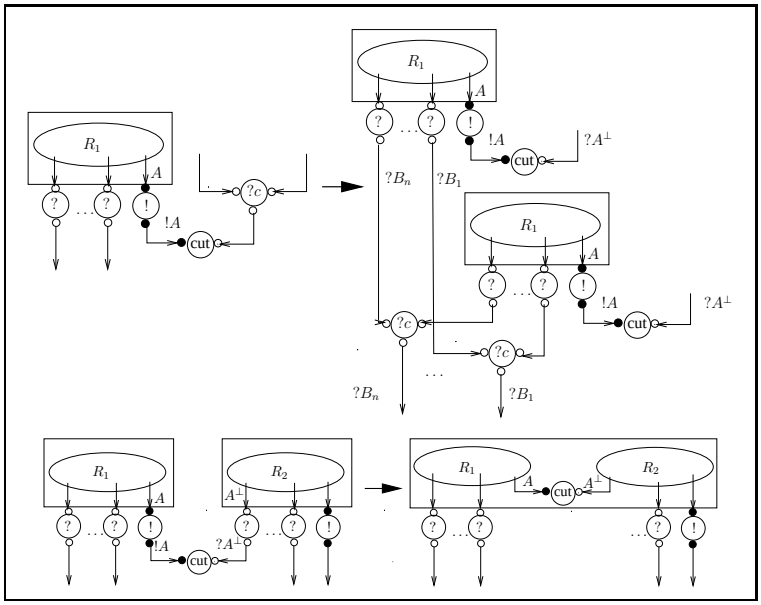

Fig. 5. Reduction steps (2/4).

- bounding the number of steps of round i by using $|R^i|_i$,
- bounding the size increase, that is to say bounding $|R^{i+1}|_{i+1}$ by using $|R^i|_i$.

Now, in the present work we will adapt essentially the same reduction strategy and methodology for obtaining the bound, with the following modifications:

- on the strategy: we will not perform the reduction until obtaining a normal form (proof-net without cut), but we will stop when we can extract the result;
- on the methodology for obtaining the bound: we will use the assumption that the proof has a conclusion $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$ and we will make a finer analysis of the size increase.

Now let us state the key Lemma that we will use:

Lemma 2 (Size bound). *Let R be a proof-net with:*

- *only exponential and weakening cuts at depth 0,*
- *k cuts at depth 0.*

Let R' be the proof-net obtained by reducing R at depth 0. Then we have:

$$|R'|_1 \leq |R|_0^k \cdot |R|_1.$$

So if we have a bound on the number k of cuts, we obtain a polynomial bound on the size of the proof-net R' after reduction at depth 0. In any case we can bound k by $|R|_0$, but then we basically recover the usual exponential bound [Gir98].

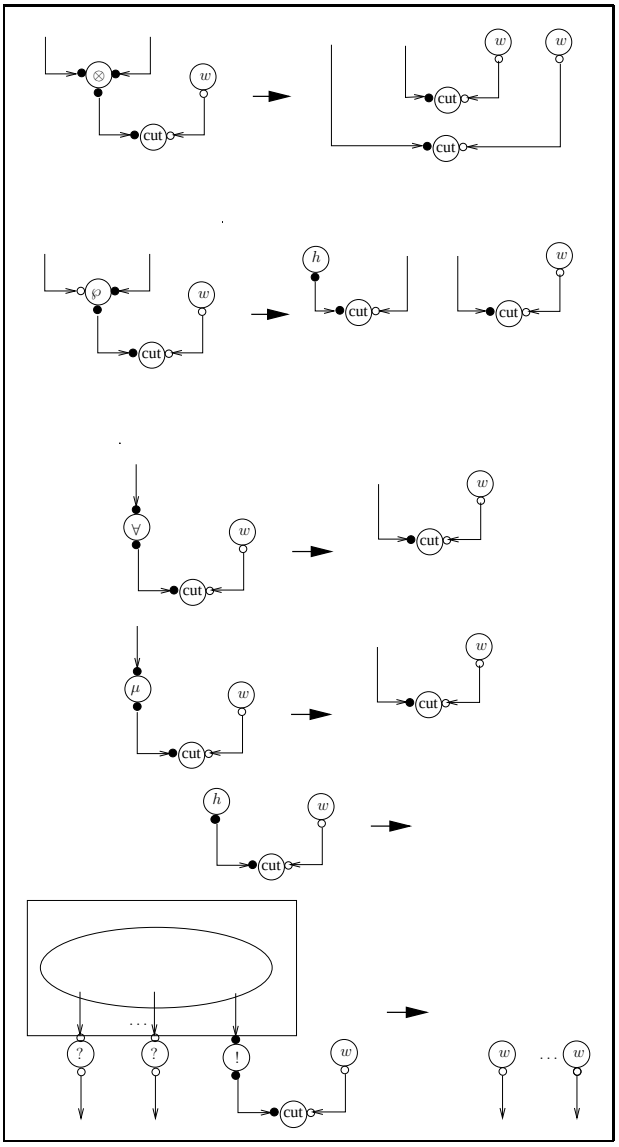

Fig. 6. Reduction steps: weakening steps (3/4).

Proof. We say a contraction node of R at depth 0 is *active* if it is above a cut node. We denote by $|R|_a$ the number of active nodes of R at depth 0. Observe that we have $|R|_a + 1 \leq |R|_0$.

We will prove the following statement by induction on k:

$$|R'|_1 \leq (|R|_a + 1)^k \cdot |R|_1.$$

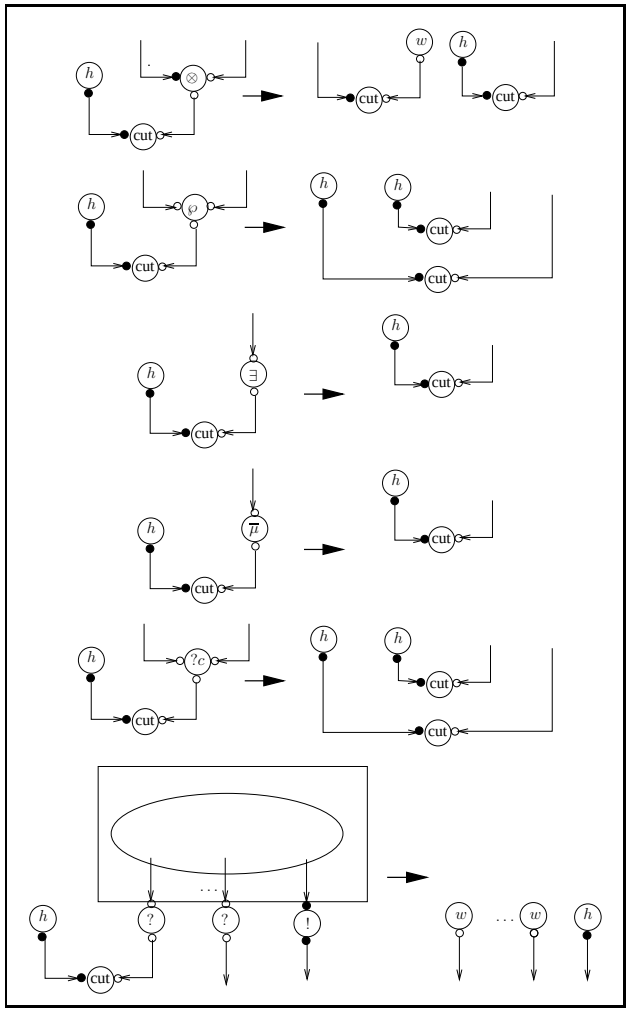

Fig. 7. Reduction steps: h-steps (4/4).

If $k = 0$ the result is trivial. Assume now the result valid for k and consider R with $k + 1$ exponential cuts at depth 0.

First let us consider the case where there is a weakening cut among these $k + 1$ cuts. We then reduce it persistently, following the weakening steps and h-steps of Fig. 6-7, and we end up with a proof-net R' such that: $|R'|_a \leq |R|_a$, $|R'|_1 \leq |R|_1$ and R' has k cuts at depth 0. We can then apply the induction hypothesis to R' and easily conclude.

Now, consider the case where we have $k + 1$ exponential cuts. Using the correctness criterion one can check (e.g. as in [BP01]) that R admits a *special cut* c, that is to say an exponential cut for which the box $\mathcal{B}$ corresponding to the

! node does not have any cut below its auxilliary ports. We completely reduce the cut c, that is to say we reduce c and hereditarily all the cuts of its exponential tree until performing box-box or axiom reduction steps. The increase of size at depth 1 is due to the duplications of the box $\mathcal{B}$, which is copied at most $|R|_a$ times. Note that no active node is created during these reduction steps, because c is a special cut. We obtain in this way a proof-net R' such that: $|R'|_1 \leq (|R|_a + 1) \cdot |R|_1$, $|R'|_a \leq |R|_a$ and R' has k cuts at depth 0.

Besides, by induction hypothesis we have that R' can be reduced to R'' which is normal at depth 0 and:

$$|R''|_1 \leq (|R'|_a + 1)^k \cdot |R'|_1.$$

Combining the inequalities we thus get:

$$|R''|_1 \leq (|R|_a + 1)^{k+1} \cdot |R|_1.$$

We conclude by using the fact that $|R|_a + 1 \leq |R|_0$. □

We will need another result:

Lemma 3 (Readback). *Let R be a proof-net of conclusion $\mathbf{B}$ which only has exponential cuts at depth 0.*

Given R, one can in constant time decide whether it reduces to true or false.

Proof. We have $\mathbf{B} = \forall\alpha.\alpha \multimap \alpha \multimap \alpha$. Consider the conclusion $\mathbf{B}$ and the axiom $\alpha^\perp, \alpha$ introducing the r.h.s. litteral α of this $\mathbf{B}$. This axiom node N is at depth 0.

Consider the second conclusion $\alpha^\perp$ of this axiom. Let us assume for a contradiction that it is above a cut formula A and call c' the corresponding cut. Now, if A was of the form $!A'$ (resp. $?A'$) for some A', then the formula A would need to be introduced by a !-node (respectively by an auxilliary door of a box), and so N would be included in a box, which would contradict the fact that it is at depth 0. Therefore A is not of the form $!A'$ or $?A'$. So c is is not an exponential cut, hence a contradiction.

So the second conclusion $\alpha^\perp$ of the axiom is not above a cut; hence it must be above a conclusion. The only conclusion of the proof-net is B, and thus this $\alpha^\perp$ litteral is one of the two occurrences of $\alpha^\perp$ in $\forall\alpha.(\alpha^\perp \wp \alpha^\perp \wp \alpha)$. If it is the leftmost (resp. rightmost) occurrence of $\alpha^\perp$ then the result of the reduction will be true (resp. false). So we do not need to reduce R to know the resulting value; this can be done in constant time. □

Finally we get:

Proposition 1 (P soundness). *Let R be a normal proof-net of conclusion $!\mathbf{W} \vdash !^2\mathbf{B}$. Then there exists a polynomial P such that:*

any proof-net obtained by cutting R with a proof-net representing a word of length n can be evaluated in time bounded by $P(n)$.

Proof. First, note that there exists a constant a such that for any n, for any binary word w of length n, w can be represented by a proof-net R_w of size $|R_w| \leq a \cdot n$.

Now, let us examine the structure of R at depth 0. If $?W^\perp$ is obtained by a weakening it is trivial. Otherwise there is an integer $k \geq 1$ and a proof-net S of conclusion $\vdash \mathbf{W}^\perp, \ldots, \mathbf{W}^\perp, !\mathbf{B}$ with k formulas $\mathbf{W}^\perp$ such that: R is obtained from S by applying a promotion box and a certain number k' of contraction rules on $?\mathbf{W}^\perp$ formulas.

Now let R_w be a proof-net representing a word w, and let T be the proof-net obtained by cutting R with a box enclosing R_w. The proof-net T can be reduced in at most $2k'$ steps (at depth 0) into a proof-net T' consisting in a box containing S cut with k copies of R_w. Therefore $|T'| \leq |R| + k \cdot |R_w|$.

Then, since $\mathbf{W} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$, by applying k quantification reduction steps and $2k$ multiplicative reduction steps (at depth 1) we get a proof-net T'' with not cut at depth 0 and only exponential and weakening cuts at depth 1. Note that there are at most $3k$ exponential cuts at depth 1 and that $|T''| \leq |T'| \leq |R| + k \cdot |R_w|$.

Now by applying Lemma 2 to T'' at depth 1, we get that by reducing T'' at depth 1 we obtain $T^{(3)}$ with no cut at depths 0, 1 and such that $|T^{(3)}|_2 \leq |T''|_1^{3k} \cdot |T''|_2 \leq (|R| + k \cdot |R_w|)^{3k+1}$. The important point to notice here is that $(3k+1)$ does not depend on n.

Finally we perform on $T^{(3)}$, at depth 2, reduction of all cuts but exponential cuts. These only make the size decrease, by Lemma 1, and so the number of steps is bounded by $|T^{(3)}|_2$. We obtain in this way a proof-net $T^{(4)}$ of conclusion $!^2 B$, which:

- does not have any cut at depths 0 and 1,
- only has exponential cuts at depth 2.

By Lemma 3, applied here at depth 2, the result can then be computed in constant time. So on the whole the computation has been carried out in a number of steps which is polynomial in $|R_w|$, hence polynomial in n.

Moreover, as the size of each intermediary proof-net in the reduction sequence is polynomially-bounded w.r.t. n, each reduction step can be performed in polynomial time (on a Turing machine), hence the whole reduction is performed in polynomial time. □

Proposition 2 (k-EXP **soundness).** *Let R be a normal proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$. Then there exists a polynomial P such that:*

any proof-net obtained by cutting R with a proof-net representing a word of length n can be evaluated in time bounded by $2_k^{P(n)}$.

Proof. We will proceed in a way generalizing the proof of Prop. 1, but we first need for that an intermediary lemma:

Lemma 4. *Let R be a normal proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ and $2 \leq i \leq k+2$. Then there exists a polynomial Q such that:*

consider S a proof-net obtained by cutting R with a proof-net representing a word of length n; then one can in at most $2_{i-2}^{Q(n)}$ steps reduce S into a proof-net S' of size inferior to $2_{i-2}^{Q(n)}$ and with not cut at depth inferior or equal to $i-1$, and only exponential cuts at depth i.

Proof. We proceed by induction on i.

In the case where $i = 2$: we proceed as in the proof of Prop. 1; the arguments used are still valid in the case of a proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ and so we obtain a bound of the form $P(n)$.

Now, assume the property is true for $i \leq k+1$ and let us show it for $i+1$. By induction hypothesis one has obtained in $2_{i-2}^{Q(n)}$ steps a proof-net S' of size inferior to $2_{i-2}^{Q(n)}$ and with not cut at depth inferior or equal to $i-1$, and only exponential cuts at depth i. We reduce the exponential cuts at depth i following the strategy of the proof of Lemma 1, that is to say by reducing special cuts. The number of steps performed at depth i is then bounded by $|S'|_i$, and at each step the size of the proof-net at depth i decreases, and its size at depth $(i+1)$ at most doubles. Let us call S'' the resulting proof-net, which does not have any cut at depth inferior or equal to i. We thus have $|S''| \leq |S'| \cdot 2^{|S'|_i} \leq 2_{i-2}^{Q(n)} \cdot 2_{i-1}^{Q(n)}$. Therefore there is a polynomial Q' such that $|S''| \leq 2_{i-1}^{Q'(n)}$. Finally we reduce all non-exponential cuts at depth i, which makes the size decrease and takes less than $|S''|$ steps. The induction hypothesis is thus valid for $i+1$ and we are done. □

Let us come back to the proof of Prop. 2. Call S the proof-net obtained by cutting R with a proof-net representing a word of length n. It has a single conclusion, of type $!^{k+2}\mathbf{B}$. Now, we apply Lemma 4 with $i = k+2$ and thus obtain in at most $2_k^{Q(n)}$ reduction steps a proof-net S' of size inferior to $2_k^{Q(n)}$, with no cut at depth inferior or equal to $k+1$, and only exponential cuts at depth $k+2$. Therefore the proof-net S' consists of $k+2$ boxes applied to a proof-net T' such that: T' has a single conclusion of type $\mathbf{B}$ and only exponential cuts at depth 0. By Lemma 3 one can in constant time decide whether T', and thus S', evaluates to true or false.

Therefore, on the whole after at most $2_k^{Q(n)}$ reduction steps one can decide the result of the evaluation of S. Moreover as $2_k^{Q(n)}$ also bounds the size of the intermediary proof-nets in the reduction sequence, this provides a bound on the time needed to evaluate S. □

4 Extensional completeness

To prove the extensional completeness results we will need another datatype using type fixpoints, that of *Scott binary words*:

$$\mathbf{W_S} = \mu\beta.\forall\alpha.(\beta \multimap \alpha) \multimap (\beta \multimap \alpha) \multimap (\alpha \multimap \alpha).$$

Scott words have already been used in several works on implicit complexity [DLB06,BT10,RV10]. One can easily define terms for the basic operations on binary words over the type $\mathbf{W_S}$:

$$\begin{array}{lll} cons_0 = \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_0\ w) & : \mathbf{W_S} \multimap \mathbf{W_S} \\ cons_1 = \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_1\ w) & : \mathbf{W_S} \multimap \mathbf{W_S} \\ nil = \lambda s_0.\lambda s_1.\lambda x.x & : \mathbf{W_S} \\ tail = \lambda w.(w\ id\ id\ nil) & : \mathbf{W_S} \multimap \mathbf{W_S} \end{array}$$

where $id = \lambda x.x$.
For this datatype we can define a term:

$$\begin{array}{l} case\ :\ \forall\alpha.(\mathbf{W_S} \multimap \alpha) \multimap (\mathbf{W_S} \multimap \alpha) \multimap \alpha \multimap (\mathbf{W_S} \multimap \alpha) \\ case = \lambda F_0.\lambda F_1.\lambda a.\lambda w.(w\ F_0\ F_1\ a) \end{array}$$

Remark 3. Actually Scott words can also be typed in elementary affine logic, without fixed points, e.g. with the following type: $\forall P.(P \multimap \forall X.((P \multimap X \multimap P) \multimap P) \multimap P)$. However it is not clear if one could define a *case* function in this setting.

We define the following type representing the configurations of a one-tape Turing machine over a binary alphabet, with n states:

$$\mathbf{Config} = \mathbf{W_S} \otimes \mathbf{B} \otimes \mathbf{W_S} \otimes \mathbf{B^n}$$

Given an element of this type: the first component represents the left part of the tape, in reverse order; the second component represents the symbol scanned by the head ; the third component represents the right part of the tape; the fourth part represents the current state.

Lemma 5. *Let q be a polynomial over one variable, with coefficients in $\mathbb{N}$. We have:*

1. *there exists a proof of $!\mathbf{N} \multimap !\mathbf{N}$ representing the function $q(n)$;*
2. *for any $k \geq 1$, there exists a proof of $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing the function $2_k^{q(n)}$.*

Proof. Actually these encodings do not use type fixpoints and can be done in elementary linear logic.

1. Recall that addition and multiplication on tally integers can both be represented by proofs of conclusions $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$. It follows that any polynomial can be represented by a proof of conclusion $!\mathbf{N} \multimap !\mathbf{N}$.
2. By using the proof for multiplication one defines a proof representing the doubling function, *double*, with type $\mathbf{N} \multimap \mathbf{N}$. By iterating this function, that is to say with the proof corresponding to the term $exp = \lambda n.(n\ double\ \underline{1})$, of type $\mathbf{N} \multimap !\mathbf{N}$, one represents the function 2^n.
   By composing k times EXP we obtain a term exp_k of type $\mathbf{N} \multimap !^k\mathbf{N}$ representing the function 2_k^n. Finally by composing the term exp_k with the term of type $!\mathbf{N} \multimap !\mathbf{N}$ representing $q(n)$, one gets a term of type $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing the function $2_k^{q(n)}$.

□

Lemma 6. *Let $\mathcal{M}$ be a one-tape deterministic Turing machine. One can define terms:*

$$\begin{aligned} \mathit{init} &: \mathbf{W_S} \multimap \mathbf{Config}, \\ \mathit{step} &: \mathbf{Config} \multimap \mathbf{Config}, \\ \mathit{accept?} &: \mathbf{Config} \multimap \mathbf{B}, \end{aligned}$$

such that:

- *given a binary word, init produces the corresponding initial configuration of the machine,*
- *the term step computes one step of the machine on a given configuration,*
- *given a configuration, the term accept? returns true (resp. false) if its state is accepting (resp. rejecting).*

The terms *init* and *accept?* are easy. The term *step* can be constructed based on the transition function of $\mathcal{M}$, by doing a case distinction, using the term *case*, as in [DLB06] (Sect.7, Lemma 4).

Remark 4. The configuration type of (Asperti Roversi 2002) in LAL does not use type fixpoints and can be directly adapted in EAL (replacing the § connective by !). Let us denote here this EAL type by $\mathbf{Config_C}$, as it is defined by using a Church encoding of binary words, while **Config** is defined by using the Scott encoding:

$$\mathbf{Config_C} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha \multimap (\alpha \otimes \alpha \otimes \mathbf{B^n})),$$

where $\mathbf{B^n}$ is for representing the state of the machine.

Note that $\mathbf{Config_C}$ allows to define a function step_C with the analogous type $\mathbf{Config_C} \multimap \mathbf{Config_C}$. However the corresponding term $\mathit{accept?}_C$ has type $\mathbf{Config_C} \multimap !\mathbf{B}$. It does not seem possible to give a term for this purpose of type $\mathbf{Config_C} \multimap \mathbf{B}$.

This is why we are considering here EAL_μ with type fixpoints, so as to be able to use Scott binary words.

Proposition 3 (Extensional completeness).

1. *Let f be a function representing a predicate of* P. *There exists a proof of conclusion $!\mathbf{W} \vdash !^2\mathbf{B}$ representing f.*
2. *Consider $k \geq 1$ and let f be a function representing a predicate of* $\mathsf{k\text{-}EXP}$. *There exists a proof of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ representing f.*

Proof. Let us first prove statement 1.

Let $\mathcal{M}$ be a one-tape deterministic machine over a binary alphabet, of polynomial time $q(n)$, computing f. We represent its configurations with the type **Config** and we know by Lemma 6 that there is:

– a proof *step* of conclusion $\mathbf{Config} \multimap \mathbf{Config}$ representing one step of execution on a configuration.
– a proof *init* of conclusion $\mathbf{W} \vdash \mathbf{Config}$ for producing the initial configuration, and a proof *accept*? of conclusion $\mathbf{Config} \vdash \mathbf{B}$ for deciding whether a configuration is accepting or not.

Moreover by Lemma 5 (1), the polynomial q can be represented by a proof of $!\mathbf{N} \multimap !\mathbf{N}$. Note also that there is a proof *length* of conclusion $\mathbf{W} \vdash \mathbf{N}$ giving the length (as a tally integer) of a word.

Now, using *step* and iteration on $\mathbf{N}$, one can define a proof of conclusion $\mathbf{N}, !\mathbf{Config} \vdash !\mathbf{Config}$, which given an integer n and a configuration c, will produce the configuration c' obtained by n computation steps starting from c. Using the representation of the polynomial q we also obtain a proof π of $!\mathbf{N}, !^2\mathbf{Config} \vdash !^2\mathbf{Config}$, which given an integer n and a configuration c, produces the configuration c' obtained by $q(n)$ computation steps starting from c.

Now, the requested proof is obtained by composing the following constructions:

– use duplication $!\mathbf{W} \multimap !\mathbf{W} \otimes !\mathbf{W}$, *length* and the coercion $\mathbf{W} \multimap !\mathbf{W}$ to obtain $!\mathbf{W} \multimap !\mathbf{N} \otimes !^2\mathbf{W}$,
– compose it with $id \otimes init$ to obtain $!\mathbf{W} \multimap !\mathbf{N} \otimes !^2\mathbf{Config}$,
– compose it with the proof π described above, to get $!\mathbf{W} \multimap !^2\mathbf{Config}$,
– finally compose with *accept*? to obtain a proof of $!\mathbf{W} \multimap !^2\mathbf{B}$.

The resulting proof represents the computation of the machine $\mathcal{M}$, hence the predicate f.

For showing statement 2. it is sufficient to adapt the previous proof by replacing the polynomial $q(n)$, bounding the time of the Turing machine, by a function $2_k^{q(n)}$, and to use Lemma 5 (2) saying that there is a proof of type $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing $2_k^{q(n)}$. □

Finally, together the results of Propositions 1, 2 and 3 establish Theorem 1.

Remark 5. Observe that proofs of type $!\mathbf{W} \vdash !^2\mathbf{W}$ do not correspond to polynomial time functions. Indeed, one can easily define a proof *wdouble* of conclusion $\mathbf{W} \multimap \mathbf{W}$ which doubles the length of a word. By using iteration on words, applied here to this *wdouble* proof, one gets a proof *wexp* of conclusion $\mathbf{W} \vdash !\mathbf{W}$, which, given a word of length n, produces a word of length 2^n. Applying the ! rule one thus obtains a proof of $!\mathbf{W} \vdash !^2\mathbf{W}$ with the same behaviour.

To characterize the complexity class FP of polynomial time functions, one could use the type $!\mathbf{W} \multimap !^2\mathbf{W_S}$, where $\mathbf{W_S}$ is the type of Scott binary words. However a drawback of this characterization is that the proofs representing these functions could not be composed, because of the mismatch on the input and output types.

Remark 6. The proof of the previous theorem could be adapted in EAL (without fixpoint) instead of EAL_μ, by using the type $\mathbf{Config_C}$ of Remark 4, instead of **Config**. But as we have the type $accept?_C : \mathbf{Config_C} \multimap !\mathbf{B}$, we would thus get in the end the type $!\mathbf{W} \multimap !^3\mathbf{B}$ for the simulation of a polynomial time machine. This in turn would not provide a P soundness result.

5 Conclusion and future work

Elementary linear logic was up to now considered as a simple variant of elementary linear logic with good structural properties but of limited interest for complexity. We have shown here that, provided one adds to it type fixpoints, it is expressive enough to characterize P, EXP and a time hierarchy inside the elementary class. An interesting feature is that this provides a single type system in which one characterizes different complexity classes with the same term calculus, simply by considering terms of different types.

Several questions remain open. Is it possible to obtain the same result without type fixpoint? Could one also characterize in this system PSPACE and other space complexity classes in a way similar to [Lei02]? It would also be interesting to examine whether one could re-prove in this purely logical framework the classical hierarchy results, like $\mathsf{P} \neq \mathsf{EXP}$, by carrying out a diagonalisation argument.

Acknowledgements. The author would like to thank Jean-Yves Girard whose initial question, whether P could be characterized in elementary linear logic, triggered this work. Thanks also to Christophe Raffalli, for useful discussions about the typing of Scott integers in system F.

This work was partially supported by project ANR-08-BLANC-0211-01 "COMPLICE".

References

[ACM04] A. Asperti, P. Coppola, and S. Martini. (Optimal) duplication is not elementary recursive. *Information and Computation*, 193:21–56, 2004.

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):1–39, 2002.

[BC92] S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BGM10] P. Baillot, M. Gaboardi, and V. Mogbil. A polytime functional language from light linear logic. In *Proceedings of European Symposium on Programming (ESOP 2010)*, volume 6012 of *LNCS*, pages 104–124. Springer, 2010.

[BMM11] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations: a way to control resources. *Theor. Comput. Sci.*, 412(25):2776–2796, 2011.

[BP01] Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001.

[BT09] P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Inf. Comput.*, 207(1):41–62, 2009. (A preliminary conference version appeared in the proceedings of LICS'04).

[BT10] A. Brunel and K. Terui. Church => Scott = Ptime: an application of resource sensitive realizability. In *Proceedings International Workshop on Developments in Implicit Computational complExity (DICE 2010)*, volume 23 of *EPTCS*, pages 31–46, 2010.

[DJ03] V. Danos and J.-B. Joinet. Linear logic & elementary time. *Information and Computation*, 183:123–137, 2003.

[DLB06] U. Dal Lago and P. Baillot. Light affine logic, uniform encodings and polynomial time. *Mathematical Structures in Computer Science*, 16(4):713–733, 2006.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.

[GMRDR08] M. Gaboardi, J.-Y. Marion, and S. Ronchi Della Rocca. A logical account of Pspace. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)*. ACM, 2008.

[GR07] M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for lambda-calculus. In *Proceedings of Computer Science Logic (CSL 2007)*, volume 4646 of *LNCS*, pages 253–267. Springer, 2007.

[HAH11] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. In *Proceedings of Symposium on Principles of Programming Languages (POPL 2011)*, pages 357–370. ACM, 2011.

[HJ06] M. Hofmann and S. Jost. Type-based amortised heap-space analysis. In *Proceedings of European Symposium on Programming (ESOP 2006)*, volume 3924 of *LNCS*, pages 22–37. Springer, 2006.

[Hof03] M. Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003.

[Jon01] N. D. Jones. The expressive power of higher-order types or, life without cons. *J. Funct. Program.*, 11(1):5–94, 2001.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoret. Comput. Sci.*, 318(1–2):163–180, 2004.

[Lam96] F. Lamarche. From proof nets to games. *Electr. Notes Theor. Comput. Sci.*, 3:107–119, 1996.

[Lei91] D. Leivant. A foundational delineation of computational feasiblity. In *Proceedings Symposium on Logic in Computer Science (LICS 91)*, pages 2–11. IEEE Computer Society, 1991.

[Lei94] D. Leivant. Predicative recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.

[Lei02] D. Leivant. Calibrating computational feasibility by abstraction rank. In *Proceedings LICS'02*, pages 345–353. IEEE Computer Society, 2002.

[RV10] L. Roversi and L. Vercelli. Safe Recursion on Notation into a Light Logic by Levels. In *Proceedings of International Workshop on Developments in Implicit Computational complexity (DICE 2010)*, volume 23 of *EPTCS*, pages 63 – 77, 2010.